A. Policy Optimization under Unknown Dynamics

The policy optimization procedure employed in this work follows the method described by Levine and Abbeel (Levine & Abbeel, 2014), which we summarize in this appendix. The aim is to optimize Gaussian trajectory distributions $q(\tau) = q(\mathbf{x}_1) \prod_t q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t) q(\mathbf{u}_t|\mathbf{x}_t)$ with respect to their expected cost $E_{q(\tau)}[c_\theta(\tau)]$. This optimization can be performed by iteratively optimizing $E_{q(\tau)}[c_\theta(\tau)]$ with respect to the linear-Gaussian conditionals $q(\mathbf{u}_t|\mathbf{x}_t)$ under a linear-Gaussian estimate of the dynamics $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. This optimization can be performed using the standard linear-quadratic regulator (LQR). However, when the dynamics of the system are not known, the linearization $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ cannot be obtained directly. Instead, Levine and Abbeel (Levine & Abbeel, 2014) propose to estimate the local linear-Gaussian dynamics $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ using samples from $q(\tau)$, which can be obtained by running the linear-Gaussian controller $q(\mathbf{u}_t|\mathbf{x}_t)$ on the physical system. The policy optimization procedure then consists of iteratively generating samples from $q(\mathbf{u}_t|\mathbf{x}_t)$, fitting $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ to these samples, and updating $q(\mathbf{u}_t|\mathbf{x}_t)$ under these fitted dynamics by using LQR.

This policy optimization procedure has several important nuances. First, the LQR update can modify the controller $q(\mathbf{u}_t|\mathbf{x}_t)$ arbitrarily far from the previous controller. However, because the real dynamics are not linear, this new controller might experience very different dynamics on the physical system than the linear-Gaussian dynamics $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ used for the update. To limit the change in the dynamics under the current controller, Levine and Abbeel (Levine & Abbeel, 2014) propose solving a modified, constrained problem for updating $q(\mathbf{u}_t|\mathbf{x}_t)$, given as following:

$$\max_{q \in \mathcal{N}} E_q[c_\theta(\tau)] \text{ s.t. } D_{\text{KL}}(q \| \hat{q}) \leq \epsilon,$$

where $\hat{q}$ is the previous controller. This constrained problem finds a new trajectory distribution $q(\tau)$ that is close to the previous distribution $\hat{q}(\tau)$, so that the dynamics violation is not too severe. The step size ϵ can be chosen adaptively based on the degree to which the linear-Gaussian dynamics are successful in predicting the current cost (Levine et al., 2015). Note that when policy optimization is interleaved with IOC, special care must be taken when adapting this step size. We found that an effective strategy was to use the step size rule described in prior work (Levine et al., 2015). This update involves comparing the predicted and actual improvement in the cost. We used the preceding cost function to measure both improvements since this cost was used to make the update.

The second nuance in this procedure is in the scheme used to estimate the dynamics $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$. Since these dynamics are linear-Gaussian, they can be estimated by solving a separate linear regression problem at each time step, using the samples gathered at this iteration. The sample complexity of this procedure scales linearly with the dimensionality of the system. However, this sample complexity can be reduced dramatically if we consider the fact that they dynamics at nearby time steps are strongly correlated, even across iterations (due to the previously mentioned KL-divergence constraint). This property can be exploited by fitting a crude global model to all of the samples gathered during the policy optimization procedure, and then using this global model as a prior for the linear regression. A good choice for this global model is a Gaussian mixture model (GMM) over tuples of the form $(\mathbf{x}_t, \mathbf{u}_t, \mathbf{x}_{t+1})$, as discussed in prior work (Levine & Abbeel, 2014). This GMM is refitted at each iteration using all available interaction data, and acts as a prior when fitting the time-varying linear-Gaussian dynamics $q(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$.

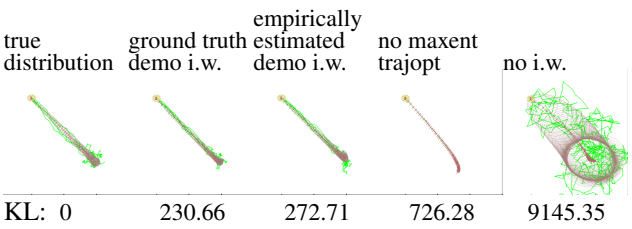

Figure 4. KL divergence between trajectories produced by our method, and various ablations, to the true distribution. Guided cost learning recovers trajectories that come close to both the mean and variance of the true distribution using 40 demonstrated trajectories, whereas the algorithm without MaxEnt policy optimization or without importance weights recovers the mean but not the variance.

B. Consistency Evaluation

We evaluated the consistency of our algorithm by generating 40 demonstrations from 4 known linear Gaussian trajectory distributions of a second order point mass system, each traveling to the origin from different starting positions. The purpose of this experiment is to verify that, in simple domains where the exact cost function can be learned, our method is able to recover the true cost function successfully. To do so, we measured the KL divergence between the trajectories produced by our method and the true distribution underlying the set of demonstrations. As shown in Figure 4, the trajectory distribution produced by guided cost learning with ground truth demo importance weights (weights based on the true distribution from which the demonstrations were sampled, which is generally unknown) comes very close to the true distribution, with a KL divergence of 230.66 summed over 100 timesteps. Empiri-

cally estimating the importance weights of the demos produces trajectories with a slightly higher KL divergence of 272.71, costing us very little in this domain. Dropping the demo and sample importance weights entirely recovers a similar mean, but significantly overestimates the variance. Finally, running the algorithm without a maximum entropy term in the policy optimization objective (see Section 4.2) produces trajectories with similar mean, but 0 variance. These results indicate that correctly incorporating importance weights into sample-based maximum entropy IOC is crucial for recovering the right cost function. This contrasts with prior work, which suggests dropping the importance weights (Kalakrishnan et al., 2013).

C. Neural Network Parametrization and Initialization

We use expressive neural network function approximators to represent the cost, using the form:

$$c_\theta(\mathbf{x}_t, \mathbf{u}_t) = \|A\mathbf{y}_t + b\|^2 + w_{\mathbf{u}}\|\mathbf{u}_t\|^2$$

This parametrization can be viewed as a cost that is quadratic in a set of learned nonlinear features $\mathbf{y}_t = f_\theta(\mathbf{x}_t)$ where f_θ is a multilayer neural network with rectifying nonlinearities of the form $\max(z, 0)$. Since simpler cost functions are generally preferred, we initialize the f_θ to be the identity function by setting the parameters of the first fully-connected layer to contain the identity matrix and the negative identity matrix (producing hidden units which are double the dimension of the input), and all subsequent layers to the identity matrix. We found that this initialization improved generalization of the learned cost.

D. Detailed Description of Task Setup

All of the simulated experiments used the MuJoCo simulation package (Todorov et al., 2012), with simulated frictional contacts and torque motors at the joints used for actuation. All of the real world experiments were on a PR2 robot, using its 7 DOF arm controlled via direct effort control. Both the simulated and real world controllers were run for 5 seconds at 20 Hz resulting in 100 time steps per rollout. We describe the details of each system below.

In all tasks except for 2D navigation (which has a small state space and complex cost), we chose the dimension of the hidden layers to be approximately double the size of the state, making it capable of representing the identity function.

2D Navigation: The 2D navigation task has 4 state dimensions (2D position and velocity) and 2 action dimensions. Forty demonstrations were generated by optimizing trajectories for 32 randomly selected positions, with at least 1 demonstration from each starting position. The neural network cost was parametrized with 2 hidden layers of dimension 40 and a final feature dimension of 20.

Reaching: The 2D reaching task has 10 dimensions (3 joint angles and velocities, 2-dimensional end effector position and velocity). Twenty demonstrations were generated by optimizing trajectories from 12 different initial states with arbitrarily chosen joint angles. The neural network was parametrized with 2 hidden layers of dimension 24 and a final feature dimension of 100.

Peg insertion: The 3D peg insertion task has 26 dimensions (7 joint angles, the pose of 2 points on the peg in 3D, and the velocities of both). Demonstrations were generated by shifting the hole within a 0.1 m $\times$ 0.1 m region on the table. Twenty demonstrations were generated from sixteen demonstration conditions. The neural network was parametrized with 2 hidden layers of dimension 52 and a final feature dimension of 100.

Dish: The dish placing task has 32 dimensions (7 joint angles, the 3D position of 3 points on the end effector, and the velocities of both). Twenty demonstrations were collected via kinesthetic teaching on nine positions along a 43 cm dish rack. A tenth position, spatially located within the demonstrated positions, was used during IOC. The input to the cost consisted of the 3 end effector points in 3D relative to the target pose (which fully define the pose of the gripper) and their velocities. The neural network was parametrized with 1 hidden layer of dimension 64 and a final feature dimension of 100. Success was based on whether or not the plate was in the correct slot and not broken.

Pouring: The pouring task has has 40 dimensions (7 joint angles and velocities, the 3D position of 3 points on the end effector and their velocities, 2 learned visual feature points in 2D and their velocities). Thirty demonstrations were collected via kinesthetic teaching. For each demonstration, the target cup was placed at a different position on the table within a 28 cm $\times$ 13 cm rectangle. The autoencoder was trained on images from the 30 demonstrations (consisting of 3000 images total). The input to the cost was the same as the state but omitting the joint angles and velocities. The neural network was parametrized with 1 hidden layer of dimension 80 and a final feature dimension of 100. To measure success, we placed 15 almonds in the grasped cup and measured the percentage of the almonds that were in the target cup after the executed motion.

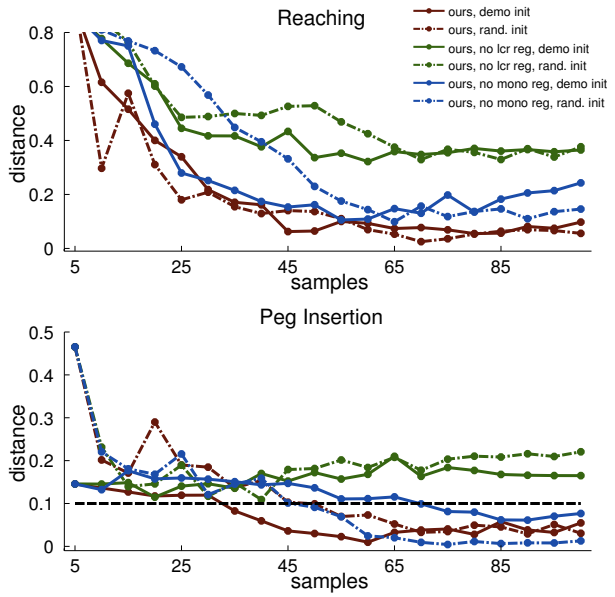

Figure 5. Comparison showing ablations of our method with leaving out one of the two regularization terms. The monotonic regularization improves performance in three of the four task settings, and the local constant rate regularization significantly improves performance in all settings. Reported distance is averaged over four runs of IOC on four different initial conditions.

E. Regularization Evaluation

We evaluated the performance with and without each of the two regularization terms proposed in Section 5 on the simulated reaching and peg insertion tasks. As shown in Figure 5, both regularization terms help performance. Notably, the learned trajectories fail to insert the peg into the hole when the cost is learned using no local constant rate regularization.